

Reactive Databases (with Java + Azure)



<https://aka.ms/reactive-java>

State Of the Reactive Java Nation

- Jakarta EE & Quarkus
- RxJava
- JDK 9+ → Reactive Streams
- Spring 5
- Spring Boot 2

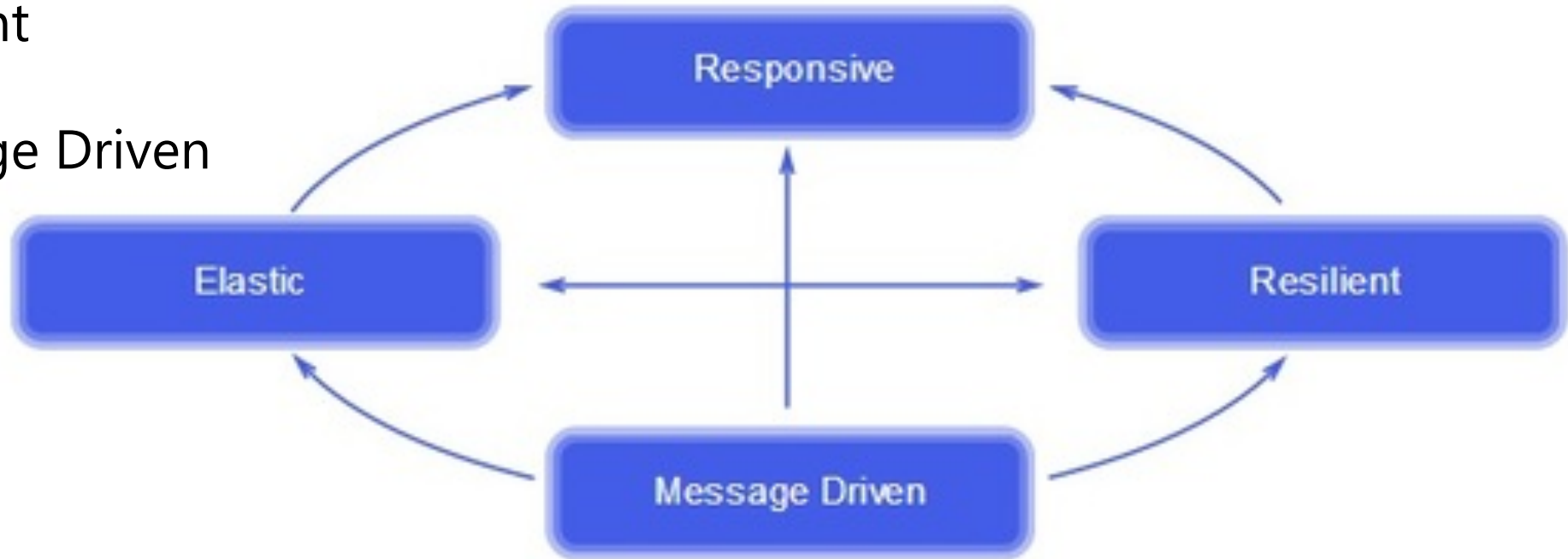
A definition

Reactive Programming is all about non-blocking applications that are asynchronous and event-driven and require a small number of threads to scale

-Spring.io

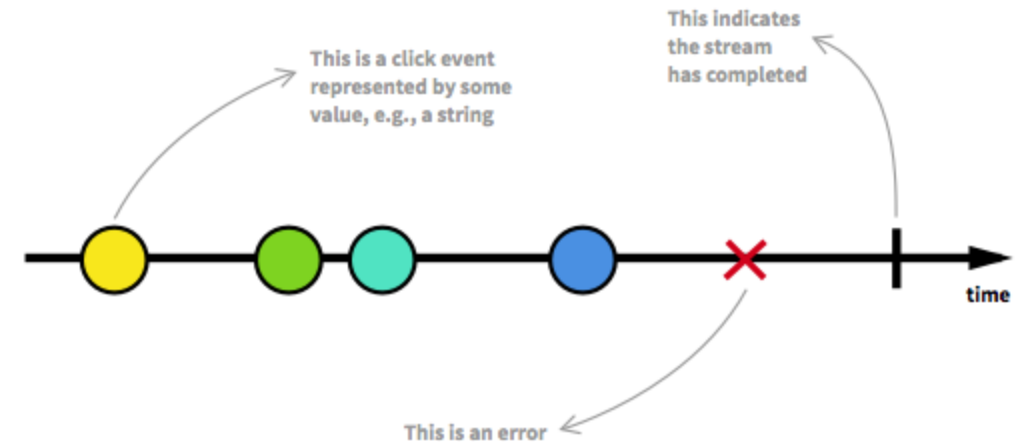
The Reactive Manifesto

- Responsive
- Resilient
- Elastic
- Message Driven



What is Reactive Programming?

- **Observer**
 - Interface to notify an object that the next item in a sequence it is watching it is available
- **Streams**
 - Controlled exchange of stream data across Applications
- **Back-pressure**
 - control the flow of a Stream between producer and consumer



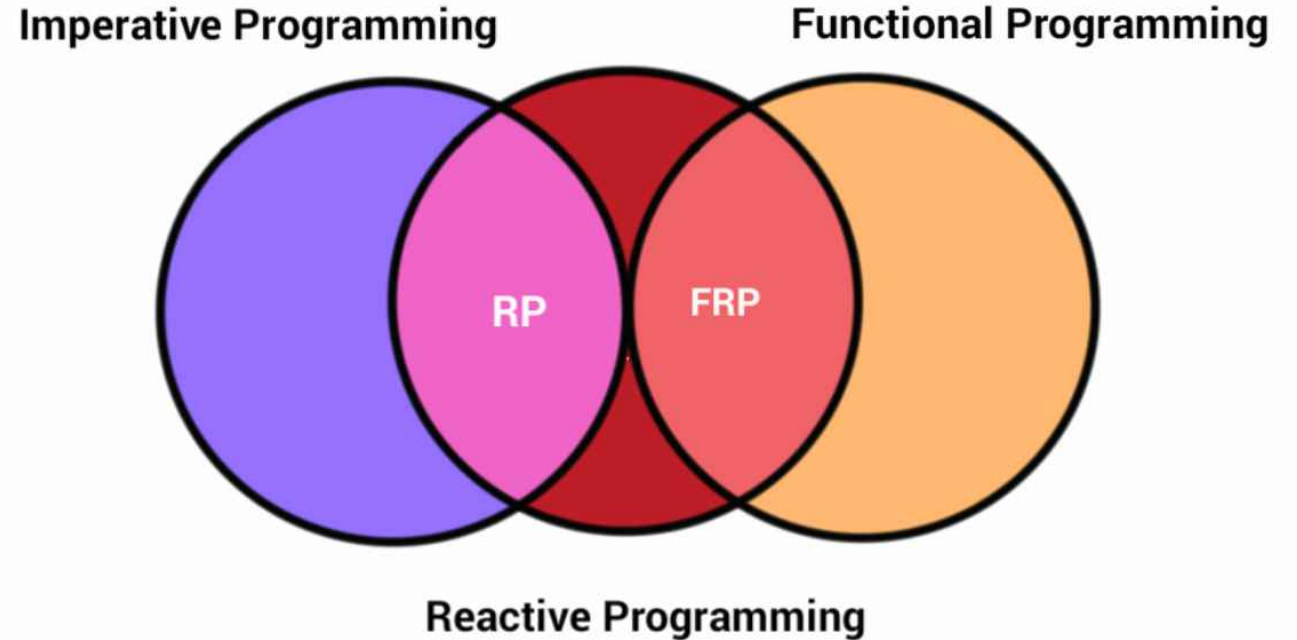
What is Functional Reactive Programming?

Conal Elliot defined FRP back in 1998, in his paper "Functional Reactive Animation":

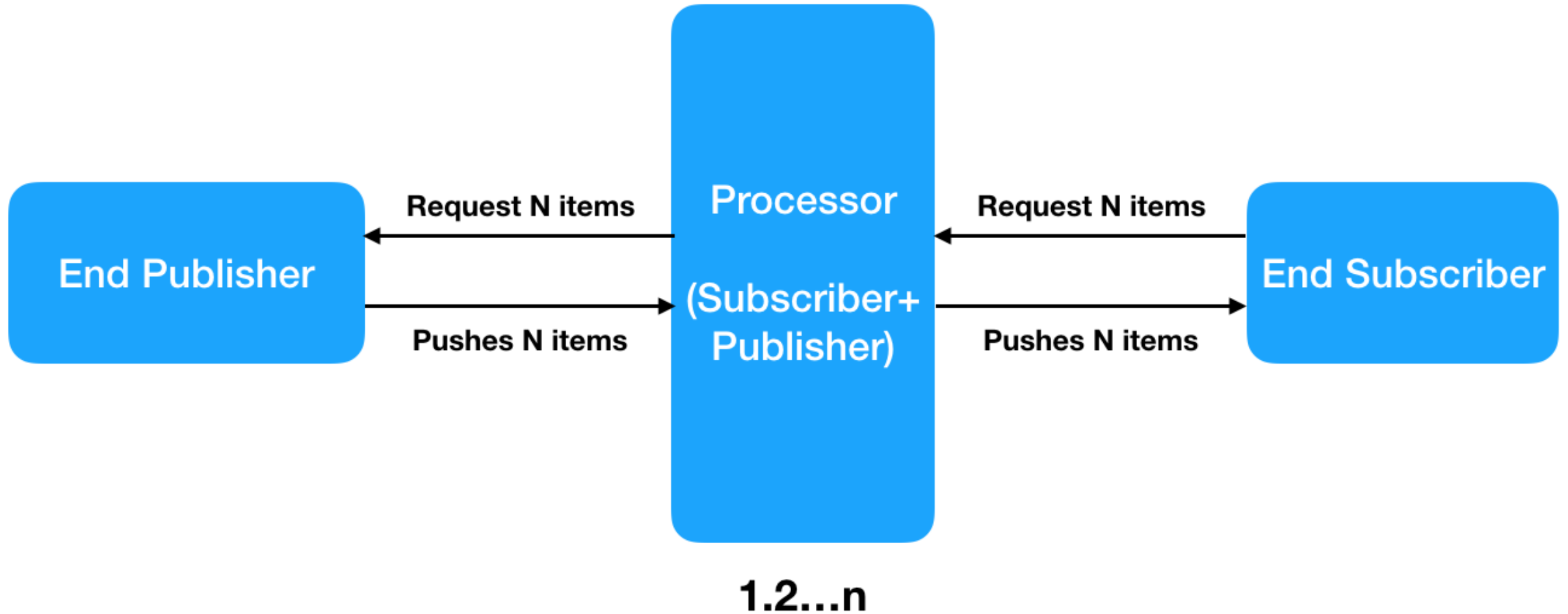
- “FRP expressions describe entire evolutions of values over time, representing these evolutions directly as first-class values

What is Functional reactive programming?

- Compositionality
 - Being able to compose functions
- Immutability
- Guarantees inherently parallelisable

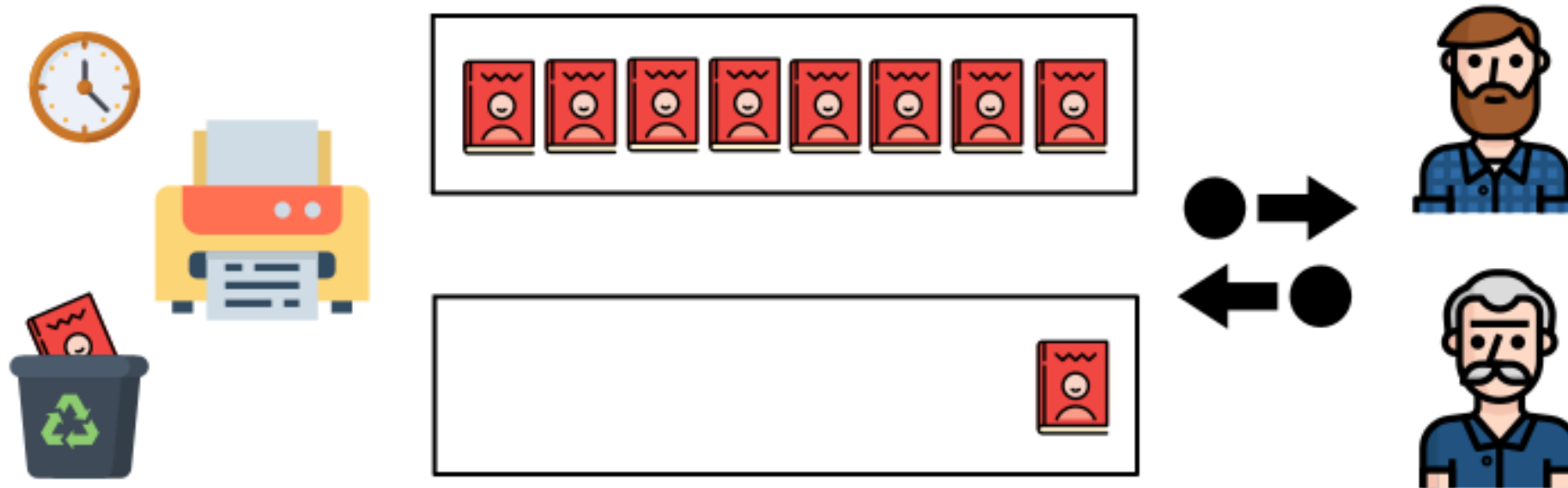


Demo Time – Reactive Basics



Demo Time – Reactive Basics

The Reactive Magazine Publisher
by ThePracticalDeveloper.com

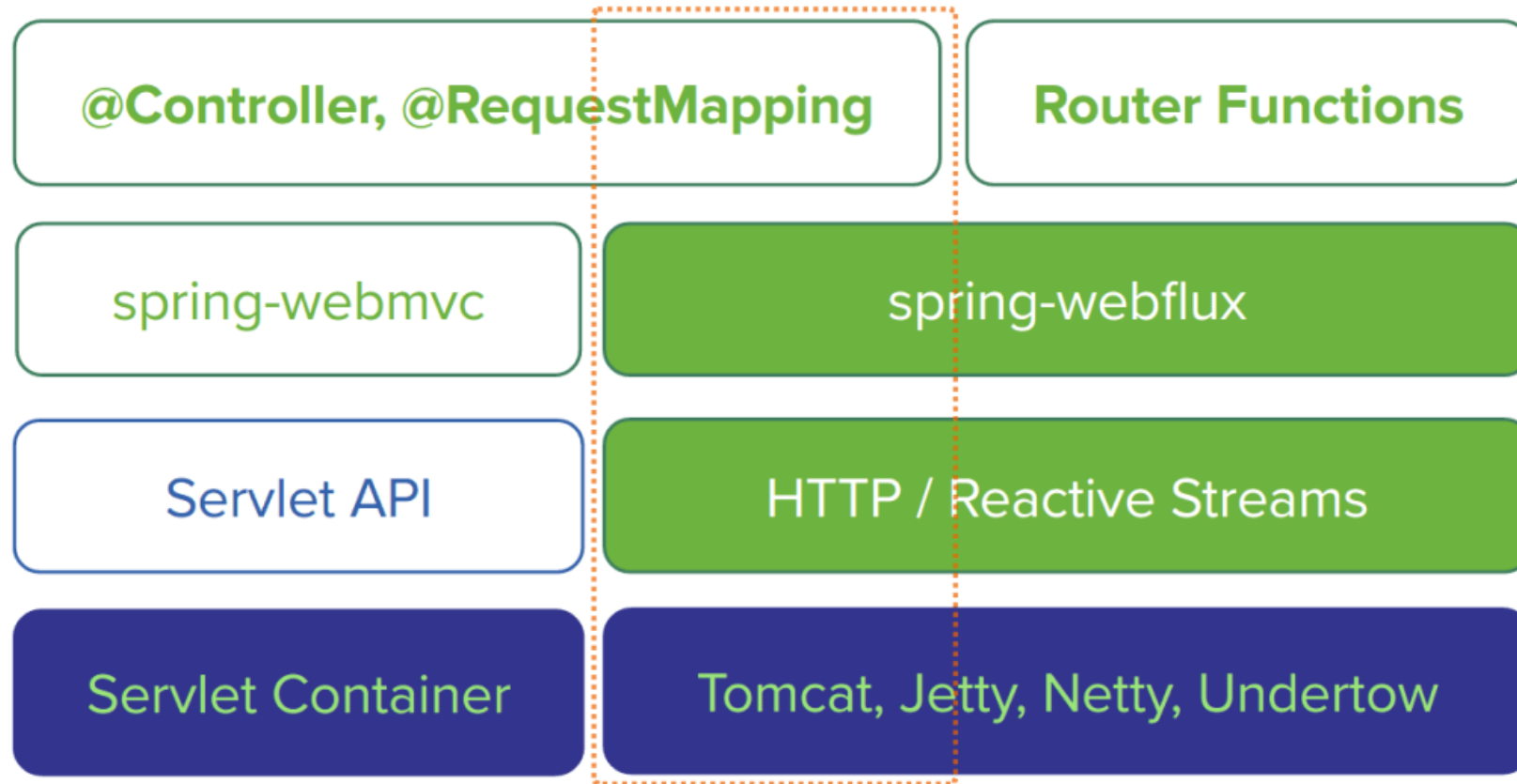


<https://github.com/reactive-book/java-9-flow>

Spring 5

- Java 8+
- Netty
- Webflux
- MVC or Roll-your own http handling

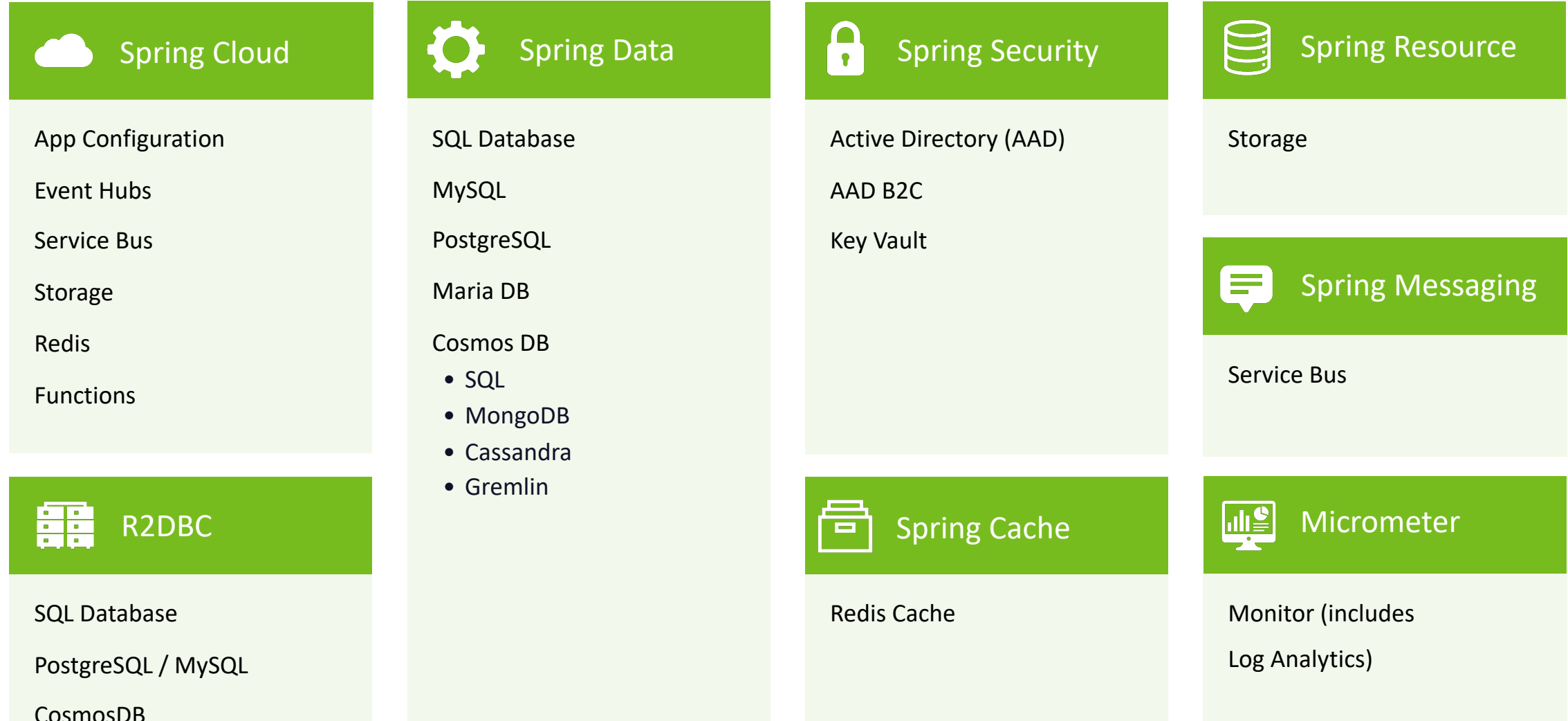
Spring Reactor



Reactive Mongo

```
public interface QuoteMongoReactiveRepository  
extends ReactiveCrudRepository<Quote, String> {  
  
}
```

Reactive Spring Data Integration with Azure



Reactive Rest

```
@GetMapping("/quotes-reactive")  
public Flux<Quote> getQuoteFlux() {  
  
    return quoteMongoReactiveRepository.findAll();  
  
}
```

Angular

```
quotes: Quote[] = new Array();
url: string = 'http://localhost:8080/quotes-reactive';

getQuoteStream(page?: number, size?: number): Observable<Array<Quote>> {
  this.quotes = new Array();
  return Observable.create((observer) => {
    let url = this.url;

    let eventSource = new EventSource(url);
    eventSource.onmessage = (event) => {
      console.debug('Received event: ', event);
      let json = JSON.parse(event.data);
      this.quotes.push(new Quote(json['id'], json['book'], json['content']));
      observer.next(this.quotes);
    };
    eventSource.onerror = (error) => observer.error('EventSource error: ' + error);
  });
}
```

Angular

```
quotes: Quote[] = new Array();
url: string = 'http://localhost:8080/quotes-reactive';

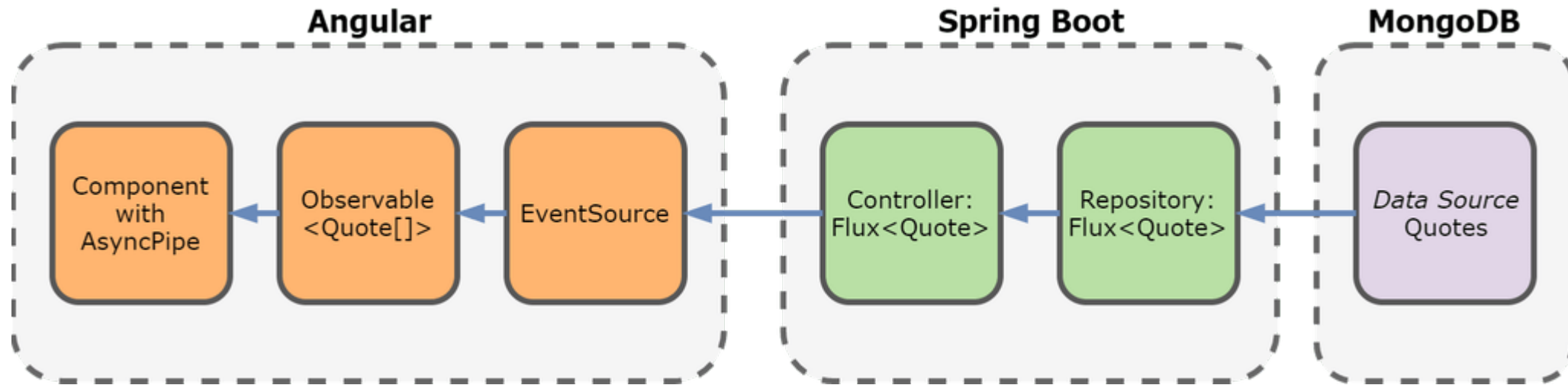
getQuoteStream(page?: number, size?: number): Observable<Array<Quote>> {
  this.quotes = new Array();
  return Observable.create((observer) => {
    let url = this.url;

    let eventSource = new EventSource(url);
    eventSource.onmessage = (event) => {
      console.debug('Received event: ', event);
      let json = JSON.parse(event.data);
      this.quotes.push(new Quote(json['id'], json['book'], json['content']));
      observer.next(this.quotes);
    };
    eventSource.onerror = (error) => observer.error('EventSource error: ' + error);
  });
}
```


How do you test?

- `curl -H "Accept: text/event-stream"` <http://localhost:8080/quotes-reactive>
- Spring WebClient
- Spring WebTestClient
- Postman? Browser?

Demo Time – Full Reactive Stack



<https://github.com/mechero/full-reactive-stack>

Pros

- **Responsive**
 - Processing in batches(back-pressure)
- **Resilient** and **Elastic**
 - back-pressure
- **Message Driven**
 - Reactor MicroQueues

Cons

- Immutability?
- Debugging complexity
- Steeeeeeeep learning curve
- Do we understand Blocking?
 - “it doesn’t matter if you use a Reactive Web approach in the backend, it won’t be really reactive and non-blocking unless your client is able to handle it as well”

Finally – Slides, Code and Resources



<https://aka.ms/reactive-java>